

ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE SETDEF

OBJECT MODULE PLACED IN SETDEF.OBJ

COMPILER INVOKED BY: PLM80 SETDEF,PLM PAGESWIDTH(132) DEBUG OPTIMIZE

1 \$ TITLE('CP/M 3.0 --- SETDEF')
 setdef:
 do;

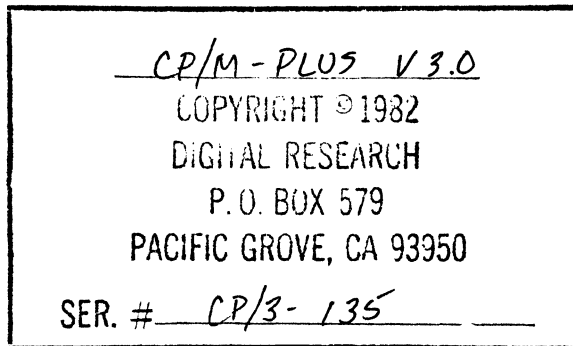
/*
 Copyright (C) 1982
 Digital Research
 P.O. Box 579
 Pacific Grove, CA 93950
*/

/*
Written: 27 July 82 by John Knight
Modified: 30 Sept 82 by Doug Huskey
Modified: 03 Dec 82 by Bruce Skidmore
*/

/*
*
* LITERALS AND GLOBAL VARIABLES *
* *
*

2 1 declare
 true literally '1',
 false literally '0',
 forever literally 'while true',
 lit literally 'literally',
 proc literally 'procedure',
 dcl literally 'declare',
 addr literally 'address',
 cr literally '13',
 tab literally '9',
 lf literally '10',
 ctrlc literally '3',
 ctrlx literally '18h',
 bksp literally '8',
 con\$width\$offset literally 'lah',
 drive0\$offset literally '4ch',
 drive1\$offset literally '4dh',
 drive2\$offset literally '4eh',
 drive3\$offset literally '4fh',
 temp\$drive\$offset literally '50h',
 ccp\$flag1\$offset literally '17h',
 ccp\$flag2\$offset literally '18h',
 pg\$mode\$offset literally '2ch',
 pg\$def\$offset literally '2dh',
 cpmversion literally '30h';

3 1 declare drive\$table (4) byte;
4 1 declare order\$table (2) byte initial(0);



```

5 1      declare drive (4) byte;
6 1      declare temp$drive byte;
7 1      declare ccp$flag1 byte;
8 1      declare ccp$flag2 byte;
9 1      declare con$width byte;
10 1     declare i byte;
11 1     declare begin$buffer address;
12 1     declare buf$length byte;

        /* display control variables */
13 1     declare show$drive  byte initial(true);
14 1     declare show$order  byte initial(true);
15 1     declare show$temp   byte initial(true);
16 1     declare show$page   byte initial(true);
17 1     declare show$display byte initial(true);

18 1     declare scbpd structure
        (offset byte,
         set   byte,
         value address);

        /* scanner variables and data */
19 1     declare
        options(*) byte data
            ('TEMPORARY'ORDER'PAGE'DISPLAY'NO'COM'SUB'NOPAGE'NODISPLAY',
            'ON'OFF',0ffh),

        options$offset(*) byte data
            (0,10,16,21,29,32,36,40,47,57,60,63),

        drives(*) byte data
            ('*A:*B:*C:*D:*E:*F:*G:*H:*I:*J:*K:*',
            'L:*M:*N:*O:*P:',0ffh),

        drives$offset(*) byte data
            (0,2,5,8,11,14,17,20,23,26,29,32,
            35,38,41,44,47,49),

        end$list   byte data (0ffh),

        delimiters(*) byte data (0,'[]=, ./;()',0,0ffh),

        SPACE   byte data(5),
        j       byte initial(0),
        buf$ptr  address,
        index   byte,
        endbuf  byte,
        delimiter byte;

20 1     declare end$of$string  byte initial ('');

21 1     declare plm label public;

        /*****
        *
        *      B D O S  INTERFACE
        *
        *****/

```

COPYRIGHT © 1982
 DIGITAL RESEARCH
 P. O. BOX 579
 PACIFIC GROVE, CA 93950

SER. # _____

```

*
*****/

22 1  mon1:
      procedure (func,info) external;
23 2      declare func byte;
24 2      declare info address;
25 2      end mon1;

26 1  mon2:
      procedure (func,info) byte external;
27 2      declare func byte;
28 2      declare info address;
29 2      end mon2;

30 1  mon3:
      procedure (func,info) address external;
31 2      declare func byte;
32 2      declare info address;
33 2      end mon3;

34 1  declare cmdrv    byte    external; /* command drive */
35 1  declare fcb (1)  byte    external; /* 1st default fcb */
36 1  declare fcb16 (1) byte    external; /* 2nd default fcb */
37 1  declare pass0    address external; /* 1st password ptr */
38 1  declare len0     byte    external; /* 1st passwd length */
39 1  declare pass1    address external; /* 2nd password ptr */
40 1  declare len1     byte    external; /* 2nd passwd length */
41 1  declare tbuff (1) byte    external; /* default dma buffer */

```

```

/*****
*
*      B D O S  Externals
*
*****/

```

```

42 1  printchar:
      procedure(char);
43 2      declare char byte;
44 2      call mon1(2,char);
45 2      end printchar;

46 1  print$buf:
      procedure (buffer$address);
47 2      declare buffer$address address;
48 2      call mon1 (9,buffer$address);
49 2      end print$buf;

50 1  version: procedure address;
      /* returns current cp/m version # */
51 2      return mon3(12,0);
52 2      end version;

53 1  getsbbyte: procedure (offset) byte;
54 2      declare offset byte;

```

COPYRIGHT ©1982
 DIGITAL RESEARCH
 P. O. BOX 579
 PACIFIC GROVE, CA 93950

SER. # _____

```

55 2      scbpd.offset = offset;
56 2      scbpd.set = 0;
57 2      return mon2(49,.scbpd);
58 2      end getschbyte;

59 1      setschbyte:
        procedure (offset,value);
60 2      declare offset byte;
61 2      declare value byte;
62 2      scbpd.offset = offset;
63 2      scbpd.set = 0ffh;
64 2      scbpd.value = double(value);
65 2      call mon1(49,.scbpd);
66 2      end setschbyte;

```

COPYRIGHT ©1982
 DIGITAL RESEARCH
 P. O. BOX 579
 PACIFIC GROVE, CA 93950

SER. # _____

```

/*****
*
*      SUBROUTINES
*
*****/

```

*** Option scanner ***

```

67 1      separator: procedure(character) byte;

                /* determines if character is a
                delimiter and which one */

68 2      declare k byte,
                character byte;

69 2      k = 1;
70 2      loop: if delimiters(k) = end$list then return(0);
72 2      if delimiters(k) = character then return(k); /* null = 25 */
74 2      k = k + 1;
75 2      go to loop;

76 2      end separator;

77 1      opt$scanner: procedure(list$ptr,off$ptr,idx$ptr);
                /* scans the list pointed at by idx$ptr
                for any strings that are in the
                list pointed at by list$ptr.
                Offptr points at an array that
                contains the indices for the known
                list. Idxptr points at the index
                into the list. If the input string
                is unrecognizable then the index is
                0, otherwise > 0.

```

First, find the string in the Known list that starts with the same first character. Compare up until the next delimiter on the input, if every input character matches then check for uniqueness. Otherwise try to find another Known string that has its first character match, and repeat. If none can be found then return invalid.

To test for uniqueness, start at the next string in the Known list and try to get another match with the input. If there is a match then return invalid.

else move pointer past delimiter and return.

P.Balme */

```
78 2  declare
      buff      based buff$ptr (1) byte,
      idx$ptr    address,
      off$ptr    address,
      list$ptr   address;
```

```
79 2  declare
      i          byte,
      j          byte,
      list       based list$ptr (1) byte,
      offsets    based off$ptr (1) byte,
      wrd$pos    byte,
      character  byte,
      letter$in$word byte,
      found$first byte,
      start      byte,
      index      based idx$ptr byte,
      save$index byte,
      (len$new, len$found) byte,
      valid      byte;
```

```
/* internal subroutines */
/* internal subroutines */
```

```
80 2  check$in$list: procedure;
      /* find Known string that has a match with
      input on the first character. Set index
      = invalid if none found. */
```

```
81 3  declare i byte;

82 3  i = start;
83 3  wrd$pos = offsets(i);
84 3  do while list(wrd$pos) <> end$list;
85 4  i = i + 1;
86 4  index = i;
```

COPYRIGHT ©1982
DIGITAL RESEARCH
P.O. BOX 579
PACIFIC GROVE, CA 93950

SER. # _____

```

87 4      if list(wrd$pos) = character then return;
89 4      wrd$pos = offsets(i);
90 4      end;
          /* could not find character */
91 3      index = 0;
92 3      return;
93 3      end check$in$list;

94 2      setup: procedure;
95 3          character = buff(0);
96 3          call check$in$list;
97 3          letter$in$word = wrd$pos;
          /* even though no match may have occurred, position
            to next input character. */
98 3          i = 1;
99 3          character = buff(1);
100 3      end setup;

101 2      test$letter: procedure;
          /* test each letter in input and known string */

102 3          letter$in$word = letter$in$word + 1;

          /* too many chars input? 0 means
            past end of known string */
103 3          if list(letter$in$word) = end$of$string then valid = false;
          else
105 3          if list(letter$in$word) <> character then valid = false;

          i = i + 1;
108 3          character = buff(i);

109 3      end test$letter;

110 2      skip: procedure;
          /* scan past the offending string;
            position buf$ptr to next string...
            skip entire offending string;
            ie., falseopt=mod, [note: comma or
            space is considered to be group
            delimiter] */
111 3          character = buff(i);
112 3          delimiter = separator(character);
          /* No skip for SETPATH */
113 3          do while ((delimiter < 1) or (delimiter > 11));
114 4              i = i + 1;
115 4              character = buff(i);
116 4              delimiter = separator(character);
117 4          end;
118 3          endbuf = i;
119 3          buf$ptr = buf$ptr + endbuf + 1;
120 3          return;
121 3      end skip;

122 2      eat$blanks: procedure;

123 3          declare charac based buf$ptr byte;

```

COPYRIGHT ©1982

DIGITAL RESEARCH

P. O. BOX 579

PACIFIC GROVE, CA 93950

SER. # _____

```

124 3      do while ((delimiter := separator(charac)) = SPACE);
125 4          buf$ptr = buf$ptr + 1;
126 4      end;

127 3      end eat$blanks;

      /*****
      /*      end of internals      */
      *****/

      /* start of procedure */

128 2      call eat$blanks;
129 2      start = 0;
130 2      call setup;

      /* match each character with the option
      for as many chars as input
      Please note that due to the array
      indices being relative to 0 and the
      use of index both as a validity flag
      and as a index into the option/mods
      list, index is forced to be +1 as an
      index into array and 0 as a flag*/

131 2      do while index <> 0;
132 3          start = index;
133 3          delimiter = separator(character);

      /* check up to input delimiter */

134 3          valid = true;      /* test$letter resets this */
135 3          do while delimiter = 0;
136 4              call test$letter;
137 4              if not valid then go to exit1;
139 4              delimiter = separator(character);
140 4          end;

141 3          go to good;

      /* input ~= this known string;
      get next known string that
      matches */

142 3      exit1:    call setup;
143 3      end;

      /* fell through from above, did
      not find a good match*/

144 2      endbuf = i;      /* skip over string & return*/
145 2      call skip;
146 2      return;

      /* is it a unique match in options
      list? */

147 2      good:    endbuf = i;
148 2      len$found = endbuf;

```

COPYRIGHT ©1982
 DIGITAL RESEARCH
 P.O. BOX 579
 PACIFIC GROVE, CA 93950

SER. # _____

```

149 2      save$index = index;
150 2      valid = false;
151 2      next$opt:
          start = index;
152 2      call setup;
153 2      if index = 0 then go to finished;

          /* look at other options and check
             uniqueness */

155 2      len$new = offsets(index + 1) - offsets(index) - 1;
156 2      if len$new = len$found then do;
158 3          valid = true;
159 3          do j = 1 to len$found;
160 4              call test$letter;
161 4              if not valid then go to next$opt;
163 4          end;
164 3      end;
165 2      else go to nextopt;

          /* fell through...found another valid
             match --> ambiguous reference */

166 2      index = 0;
167 2      call skip;      /* skip input field to next delimiter*/
168 2      return;

169 2      finished:      /* unambiguous reference */
          index = save$index;
170 2      buf$ptr = buf$ptr + endbuf;
171 2      call eat$blanks;
172 2      if delimiter <> 0 then
173 2          buf$ptr = buf$ptr + 1;
          else
174 2          delimiter = 5;
175 2      return;

176 2      end opt$scanner;

/* ----- */

177 1      crlf:  proc;
178 2          call printchar(cr);
179 2          call printchar(lf);
180 2          end crlf;

/* ----- */

/* The error processor. This routine prints the command line
   with a caret '^' under the offending delimiter, or sub-string.
   The code passed to the routine determines the error message
   to be printed beneath the command string. */

181 1      error: procedure (code);
182 2          declare (code,i,j,nlines,rem) byte;
183 2          declare (string$ptr,tstring$ptr) address;
184 2          declare chr1 based string$ptr byte;
185 2          declare chr2 based tstring$ptr byte;
186 2          declare caret$flag byte;

```

COPYRIGHT © 1982
 DIGITAL RESEARCH
 P.O. BOX 579
 PACIFIC GROVE, CA 93950

SER. # _____


```

187 2  print$command: procedure (size);
188 3  declare size byte;
189 3  do j=1 to size; /* print command string */
190 4      call printchar(chr1);
191 4      string$ptr = string$ptr + 1;
192 4  end;
193 3  call crlf;
194 3  do j=1 to size; /* print carot if applicable */
195 4      if .chr2 = buf$ptr then do;
197 5          carot$flag = true;
198 5          call printchar('^');
199 5      end;
      else
200 4          call printchar(' ');
201 4          tstring$ptr = tstring$ptr + 1;
202 4      end;
203 3  call crlf;
204 3  end print$command;

205 2  carot$flag = false;
206 2  string$ptr, tstring$ptr = begin$buffer;
207 2  con$width = getscbbyte(con$width$offset);
208 2  if con$width < 40 then con$width = 40;
210 2  nlines = buf$length / con$width; /* num lines to print */
211 2  rem = buf$length mod con$width; /* num extra chars to print */
212 2  if ((code = 1) or (code = 5)) then /* adjust carot pointer */
213 2      buf$ptr = buf$ptr - 1; /* for delimiter errors */
      else
214 2      buf$ptr = buf$ptr - endbuf - 1; /* all other errors */
215 2  call crlf;
216 2  do i=1 to nlines;
217 3      tstring$ptr = string$ptr;
218 3      call print$command(con$width);
219 3  end;
220 2  call print$command(rem);
221 2  if carot$flag then
222 2      call print$buf(('.Error at the '^'; $'));
      else
223 2      call print$buf(('.Error at end of line; $'));
224 2  if con$width < 65 then
225 2      call crlf;
226 2  do case code;
227 3      call print$buf(('.More than four drives specified$'));
228 3      call print$buf(('.Invalid delimiter$'));
229 3      call print$buf(('.Invalid drive$'));
230 3      call print$buf(('.Invalid type for ORDER option$'));
231 3      call print$buf(('.Invalid option$'));
232 3      call print$buf(('.End of line expected$'));
233 3      call print$buf(('.Drive defined twice in search path$'));
234 3      call print$buf(('.Invalid ORDER specification$'));
235 3      call print$buf(('.Must be ON or OFF$'));
236 3  end;
237 2  call crlf;
238 2  call mon1(0,0);
239 2  end error;

```

COPYRIGHT ©1982
 DIGITAL RESEARCH
 P.O. BOX 579
 PACIFIC GROVE, CA 93950

SER. # _____

```

/* ----- */

/* This is the main screen display for SETPATH. After every
successful operation, this procedure will be called to
show the results. This routine is also called whenever the
user just types SETPATH with no options. */

240 1  display$path: procedure;
241 2  declare i byte;
242 2  declare (display$flag,pg$mode,order) byte;

/* GET SETTINGS FROM SYSTEM CONTROL BLOCK */
243 2  drive(0) = getscbbyte(drive0$offset);
244 2  drive(1) = getscbbyte(drive1$offset);
245 2  drive(2) = getscbbyte(drive2$offset);
246 2  drive(3) = getscbbyte(drive3$offset);
247 2  temp$drive = getscbbyte(temp$drive$offset);
248 2  pg$mode = getscbbyte(pg$mode$offset);
249 2  ccp$flag2 = getscbbyte(ccp$flag2$offset);
250 2  display$flag = ccp$flag2 and 00$000$011b;
251 2  order = shr((ccp$flag2 and 00$011$000b),3);
/* 0 = COM, 1 = COM,SUB, 2 = SUB,COM */

/* DRIVE SEARCH PATH */
252 2  if show$drive then do;
254 3  call crlf;
255 3  call print$buf(,('Drive Search Path:',crlf,$'));
256 3  i = 0;
257 3  do while ((drive(i) <> 0ffh) and (i < 4));
258 4  call printchar(i + '1');
259 4  do case i;
260 5  call print$buf(,('st$'));
261 5  call print$buf(,('nd$'));
262 5  call print$buf(,('rd$'));
263 5  call print$buf(,('th$'));
264 5  end;
265 4  call print$buf(,(' Drive - $'));
266 4  if drive(i) = 0 then
267 4  call print$buf(,('Default$'));
268 4  else do;
269 5  call printchar(drive(i) + 40h);
270 5  call printchar('$');
271 5  end;
272 4  call crlf;
273 4  i = i + 1;
274 4  end;
275 3  end;

/* PROGRAM vs. SUBMIT SEARCH ORDER */
276 2  if show$order then do;
278 3  call crlf;
279 3  call print$buf(,('Search Order - $'));
280 3  do case order;
281 4  call print$buf(,('COM$'));
282 4  call print$buf(,('COM, SUB$'));
283 4  call print$buf(,('SUB, COM$'));
284 4  end;

```

COPYRIGHT © 1982
 DIGITAL RESEARCH
 P.O. BOX 579
 PACIFIC GROVE, CA 93950

SER. # _____

```

285 3      end;

      /* TEMPORARY FILE DRIVE */
286 2      if show$temp then do;
288 3          call crlf;
289 3          call print$buf(,('Temporary Drive    - $'));
290 3          if temp$drive > 16
              then temp$drive = 0;
292 3          if temp$drive = 0 then
293 3              call print$buf(,('Default$'));
294 3          else do;
295 4              call printchar(temp$drive + 40h);
296 4              call printchar('$');
297 4          end;
298 3      end;

```

```

      /* CONSOLE PAGE MODE */
299 2      if show$page then do;
301 3          call crlf;
302 3          call print$buf(,('Console Page Mode    - $'));
303 3          if pg$mode = 0 then
304 3              call print$buf(,('On$'));
              else
305 3              call print$buf(,('Off$'));
306 3          end;

```

```

      /* PROGRAM NAME & DRIVE DISPLAY */
307 2      if show$display then do;
309 3          call crlf;
310 3          call print$buf(,('Program Name Display - $'));
311 3          if display$flag = 0 then
312 3              call print$buf(,('Off$'));
              else
313 3              call print$buf(,('On$'));
314 3          end;
315 2      call crlf;
316 2      end display$path;

```

```

/* ----- */

```

```

/* This routine processes the search drives string. When called
this routine scans the command line expecting a drive name, a:-p;.
It puts the drive code in a drive table and continues the scan
collecting drives until more than 4 drives are specified (an error)
or an eoln or the delimiter '[' is encountered. Next it modifies
the SCB searchchain bytes so that it reflects the drive order as
inputted. No check is made to insure that the drive specified is
a known drive to the particular system being used.      */

```

```

317 1      process$drives: procedure;
318 2          declare (i,ct) byte;
319 2          show$drive = true;
320 2          index = 0;
321 2          delimiter = 0;
322 2          do i=0 to 3; /* clear drive table */
323 3              drive$table(i) = 0ffh;
324 3          end;

```

COPYRIGHT © 1982
 DIGITAL RESEARCH
 P.O. BOX 579
 PACIFIC GROVE, CA 93950

SER. # _____

```

325 2      ct = 0;
326 2      do while ((delimiter <> 1) and (delimiter <> 11)); /* not eoln */
327 3          call opt$scanner(.drives(0),.drives$offset(0),.index);
328 3          if ct > 3 then /* too many drives */
329 3              call error(0);
330 3          if index = 0 then /* invalid drive */
331 3              call error(2);
332 3          do i=0 to 3;
333 4              if drive$table(i) = (index-1) then
334 4                  call error(6); /* Drive already defined */
335 4              end;
336 3          drive$table(ct) = index-1;
337 3          ct = ct + 1;
338 3      end;
339 2      do i=0 to 3; /* update scb drive table */
340 3          call setscbbyte(drive0$offset+i,drive$table(i));
341 3      end;
342 2      end process$drives;

```

COPYRIGHT © 1982
DIGITAL RESEARCH
P.O. BOX 579
PACIFIC GROVE, CA 93950

SER. # _____

/* ----- */

/* This routine does all the processing for the options. Ie. any string beginning with a '['. The routine will handle basically five options: Temporary, Order, Display, Page, No Display and No Page. Each routine is fairly short and can be found as a branch in the case statement.
*/

```

343 1      process$options: procedure;
344 2          declare next$delim based buf$ptr byte;
345 2          declare (first$sub,paren,val) byte;
346 2          do while (delimiter <> 2) and (delimiter <> 11);
347 3              index = 0;
348 3              delimiter = 1;
349 3              call opt$scanner(.options(0),.options$offset(0),.index);
350 3              do case index;

351 4                  call error(4); /* not in options list (INVALID) */

352 4              do; /* temporary drive option */
353 5                  show$temp = true;
354 5                  if delimiter <> 3 then /* = */
355 5                      call error(1);
356 5                  call opt$scanner(.drives(0),.drives$offset(0),.index);
357 5                  if index = 0 then
358 5                      call error(2);
359 5                  call setscbbyte(temp$drive$offset,index-1);
360 5              end;

361 4              do; /* order option */
362 5                  show$order = true;
363 5                  first$sub,paren = false;
364 5                  if delimiter <> 3 then /* = */
365 5                      call error(1);
366 5                  do while ((next$delim = ' ') or (next$delim = tab)); /* skip spaces */
367 6                      buf$ptr = buf$ptr + 1;
368 6                  end;

```

```

369 5      if next$delim = '(' then do;
371 6          paren = true;
372 6          buf$ptr = buf$ptr + 1;
373 6      end;
374 5      call opt$scanner(.options(0),.options$offset(0),.index);
375 5      if ((index <> 6) and (index <> 7)) then
376 5          call error(3);
377 5      if index = 7 then /* note that the first entry was SUB */
378 5          first$sub = true;
379 5      order$table(0) = index - 6;
380 5      if (first$sub and ((delimiter = 10) or not paren)) then
381 5          call error(7); /* (SUB) not allowed */
382 5      if (delimiter <> 10) and paren then do;
384 6          call opt$scanner(.options(0),.options$offset(0),.index);
385 6          if ((index <> 6) and (index <> 7)) then
386 6              call error(3);
387 6          order$table(1) = index - 6;
388 6          if (first$sub and (index = 7)) then /* can't have SUB,SUB */
389 6              call error(7);
390 6      end;
391 5      ccp$flag2 = getschbyte(ccp$flag2$offset);
392 5      if order$table(0) = 0 then
393 5          ccp$flag2 = ccp$flag2 and 111$0$111b;
394 5      else
395 5          ccp$flag2 = ccp$flag2 or 000$1$0000b;
396 5      if order$table(1) = 0 then
397 5          ccp$flag2 = ccp$flag2 and 1111$0$111b;
398 5      else
399 5          ccp$flag2 = ccp$flag2 or 0000$1$000b;
400 5      call setscbbyte(ccp$flag2$offset,ccp$flag2);
401 5      if paren then do;
402 6          if delimiter <> 10 then
403 6              call error(1);
404 6          else
405 6              buf$ptr = buf$ptr + 1;
406 6          end;
407 5      else if delimiter = 10 then
408 5          call error(1);
409 5      if next$delim = ']' or next$delim = 0 then /* two delimiters */
410 5          delimiter = 11; /* eoln, so exit loop */
411 5      end;

/* PAGE Option */
412 4      do;
413 5          show$page = true;
414 5          val = 0;
415 5          if delimiter = 3 then do; /* = */
416 6              call opt$scanner(.options(0),.options$offset(0),.index);
417 6              if index <> 10 then
418 6                  if index = 11 then
419 6                      val = 0ffh;
420 6                  else
421 6                      call error(8);
422 6              end;
423 5          call setscbbyte(pg$mode$offset,val);
424 5          call setscbbyte(pg$def$offset,val);
425 5      end;

```

COPYRIGHT ©1982

DIGITAL RESEARCH

P.O. BOX 579

PACIFIC GROVE, CA 93950

SER. # _____

```

/* call error(4); page option now an error */

424 4      do; /* DISPLAY option */
425 5          show$display, val = true;
426 5          if delimiter = 3 then do; /* = */
428 6              call opt$scanner(.options(0), .options$offset(0), .index);
429 6          if index <> 10 then
430 6              if index = 11 then
431 6                  val = false;
                  else
432 6                      call error(8);
433 6          end;
434 5          ccp$flag2 = getscbbyte(ccp$flag2$offset);
435 5          if val then
436 5              ccp$flag2 = ccp$flag2 or 00000$0$11b; /* set bits */
                  else
437 5              ccp$flag2 = ccp$flag2 and 11111$1$00b; /* clear bits */
438 5          call setscbbyte(ccp$flag2$offset, ccp$flag2);
439 5          end;

/* call error(4); Display option now an error */

440 4      do; /* NO keyword */
441 5          call opt$scanner(.options(0), .options$offset(0), .index);
442 5          if (index <> 3) and (index <> 4) then
443 5              call error(4);
444 5          if index = 3 then do; /* NO PAGE option */
446 6              show$page = true;
447 6              call setscbbyte(pg$mode$offset, 0FFh);
448 6              call setscbbyte(pg$def$offset, 0FFh);
449 6          end;
450 5          else do; /* NO DISPLAY option */
451 6              show$display = true;
452 6              ccp$flag2 = getscbbyte(ccp$flag2$offset);
453 6              ccp$flag2 = ccp$flag2 and 11111$1$00b; /* clear bits */
454 6              call setscbbyte(ccp$flag2$offset, ccp$flag2);
455 6          end;
456 5          end;

/* call error(4); NO keyword is now an error */

457 4      call error(4); /* COM is not an option */

458 4      call error(4); /* SUB is not an option */

/* NOPAGE option */
459 4      do;
460 5          show$page = true;
461 5          call setscbbyte(pg$mode$offset, 0FFh);
462 5          call setscbbyte(pg$def$offset, 0FFh);
463 5          end;

/* NODISPLAY option */
464 4      do;
465 5          show$display = true;
466 5          ccp$flag2 = getscbbyte(ccp$flag2$offset);

```

COPYRIGHT ©1982
 DIGITAL RESEARCH
 P. O. BOX 579
 PACIFIC GROVE, CA 93950

SER. # _____

```

467 5      ccp$flag2 = ccp$flag2 and 11111$1$00b; /* clear bits */
468 5      call setscbbyte(ccp$flag2$offset,ccp$flag2);
469 5      end;

470 4      call error(4);      /* ON is not an option */

471 4      call error(4);      /* OFF is not an option */
472 4      end;
473 3      end;
474 2      end process$options;

```

```
/* - - - - - */
```

```

475 1      input$found: procedure (buffer$adr) byte;
476 2          declare buffer$adr address;
477 2          declare char based buffer$adr byte;
478 2          do while (char = ' ') or (char = 9); /* tabs & spaces */
479 3              buffer$adr = buffer$adr + 1;
480 3          end;
481 2          if char = 0 then /* eoln */
482 2              return false; /* input not found */
          else
483 2              return true; /* input found */
484 2          end input$found;

```

```
/* - - - - - */
```

```

/*****
*
*      M A I N   P R O G R A M
*
*****/

```

```

485 1      plm:
          do;
486 2          if (low(version) < cpmversion) or (high(version) = 1) then do;
488 3              call print$buf(,('Requires CP/M 3.0$'));
489 3              call monl(0,0);
490 3          end;
491 2          if not input$found(,tbuf(1)) then do;
          /* SHOW DEFAULTS */
493 3              call display$path;
494 3              call monl(0,0);      /* & terminate */
495 3          end;

```

```

          /* SET DEFAULTS */
496 2          i = 1;      /* skip over leading spaces */
497 2          do while (tbuf(i) = ' ');
498 3              i = i + 1;
499 3          end;
500 2          show$drive,show$order,show$temp,show$page,show$display
              = false;
501 2          begin$buffer = ,tbuf(1); /* note beginning of input */
502 2          buf$length = tbuf(0); /* note length of input */
503 2          buf$ptr = ,tbuf(i); /* set up for scanner */
504 2          if tbuf(i) = '[' then do; /* options, no drives */
506 3              buf$ptr = buf$ptr + 1; /* skip over '[' */

```

COPYRIGHT ©1982
 DIGITAL RESEARCH
 P. O. BOX 579
 PACIFIC GROVE, CA 93950

SER. # _____

```
507 3      call process$options;
508 3      end;
509 2      else do;          /* drives first, maybe options too */
510 3          call process$drives;
511 3          if delimiter = 1 then /* options, because we found an '[' */
512 3              call process$options;
513 3      end;
514 2      call display$path;    /* show results */
515 2      call moni(0,0);      /* & terminate */
516 2      end;
517 1      end setdef;
```

MODULE INFORMATION:

CODE AREA SIZE = 0E69H 3689D
VARIABLE AREA SIZE = 0052H 82D
MAXIMUM STACK SIZE = 000CH 12D
860 LINES READ
0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION

COPYRIGHT ©1982
DIGITAL RESEARCH
P.O. BOX 579
PACIFIC GROVE, CA 93950

SER. # _____

0000 SETDEF
 0000 MCD80A
 0004 BDISK 0080 BUFF 0080 BUFFA 0050 CMDRV 0000 CPU
 007C CR 006D DOLLA 005C FCB 006C FCB16 005C FCBA
 005C IFCB 005C IFCBA 0003 IOBYTE 0053 LENO 0056 LEN1
 0006 MAXB 0006 MEMSIZ 0005 MON1 0005 MON2 0005 MON2A
 0005 MON3 0000 OFFSET 006E PARMA 0051 PASS0 0054 PASS1
 007F RO 007D RR 007D RRECA 007F RRECO 005C SFCB
 0080 TBUFF
 0000 SETDEF
 10D6 MEMORY 1084 DRIVETABLE 1088 ORDERTABLE 108A DRIVE 108E TEMPDRIVE
 108F CCPFLAG1 1090 CCPFLAG2 1091 CONWIDTH 1092 I
 1093 BEGINBUFFER 1095 BUFLNGTH 1096 SHOWDRIVE 1097 SHOWORDER
 1098 SHOWTEMP 1099 SHOWPAGE 109A SHOWDISPLAY 109B SCBPD
 0180 OPTIONS 01C0 OPTIONSOFFSET 01CC DRIVES
 01FE DRIVESOFFSET 0210 ENDLIST 0211 DELIMITERS 021E SPACE
 109F J 10A0 BUFPTR 10A2 INDEX 10A3 ENDBUF 10A4 DELIMITER
 10A5 ENDOFSTRING 03FB PLM 04AF PRINTCHAR 10A6 CHAR
 04BF PRINTBUF 10A7 BUFFERADDRESS 04CF VERSION 04D8 GETSCBBYTE
 10A9 OFFSET 04F0 SETSCBBYTE 10AA OFFSET 10AB VALUE 0512 SEPARATOR
 10AC CHARACTER 10AD K 051B LOOP 0549 OPTSCANNER 10AE LISTPTR
 10B0 OFFPTR 10B2 IDXPTR 10B4 I 10B5 J 10B6 WRDPOS
 10B7 CHARACTER 10B8 LETTERINWORD 10B9 FOUNDFIRST 10BA START
 10BB SAVEINDEX 10BC LENNEW 10BD LENFOUND 10BE VALID
 066F CHECKINLIST 10BF I 06C5 SETUP 06E3 TESTLETTER
 0723 SKIP 077F EATBLANKS 05A9 EXIT1 05B9 GOOD 05CE NEXTOPT
 063F FINISHED 079B CRLF 07A6 ERROR 10C0 CODE 10C1 I
 10C2 J 10C3 NLINES 10C4 REM 10C5 STRINGPTR 10C7 TSTRINGPTR
 10C9 CAROTFLAG 08EB PRINTCOMMAND 10CA SIZE
 0956 DISPLAYPATH 10CB I 10CC DISPLAYFLAG
 10CD PGHODE 10CE ORDER 0B27 PROCESSDRIVES 10CF I
 10D0 CT 0C04 PROCSOPTIONS 10D1 FIRSTSUB 10D2 PAREN
 10D3 VAL 0FB2 INPUTFOUND 10D4 BUFFERADR

COPYRIGHT ©1982

DIGITAL RESEARCH

P.O. BOX 579

PACIFIC GROVE, CA 93950

SER. # _____

0000 SETDEF#
0000 MCD80A#
0000 SETDEF#

04AF 42	04B3 44	04BE 45	04BF 46	04C5 48
04CE 49	04CF 50	04CF 51	04D8 52	04D8 53
04DC 55	04E2 56	04E7 57	04F0 58	04F0 59
04F6 62	04FC 63	0501 64	0509 65	0511 66
0512 67	0516 69	051B 70	052A 71	052D 72
053D 73	0541 74	0545 75	0548 76	0549 77
066F 80	066F 82	0675 83	0680 84	0691 85
0695 86	069C 87	06AD 88	06AE 89	06BC 90
06BF 91	06C4 92	06C5 93	06C5 94	06C5 95
06CC 96	06CF 97	06D5 98	06DA 99	06E2 100

06E3 101	06E3 102	06E7 103	06F5 104	06FD 105
070E 106	0713 107	0717 108	0722 109	0723 110
0723 111	0731 112	073B 113	0750 114	0754 115
075F 116	0769 117	076C 118	0772 119	077E 120
077F 121	077F 122	077F 124	0790 125	0797 126
079A 127	055A 128	055D 129	0562 130	0565 131
056E 132	0575 133	057F 134	0584 135	058C 136
058F 137	0596 138	0599 139	05A3 140	05A6 141
05A9 142	05AC 143	05AF 144	05B5 145	05B8 146
05B9 147	05BF 148	05C2 149	05C9 150	05CE 151
05D5 152	05D8 153	05E1 154	05E4 155	0600 156
0608 158	060D 159	061C 160	061F 161	0626 162
0629 163	0630 164	0633 165	0636 166	063B 167
063E 168	063F 169	0646 170	0654 171	0657 172
065F 173	0669 174	066E 175	066F 176	079B 177
079B 178	07A0 179	07A5 180	07A6 181	08EB 187
08EF 189	08FE 190	0905 191	090C 192	0913 193
0916 194	0925 195	0932 197	0937 198	093C 199
093F 200	0944 201	094B 202	0952 203	0955 204
07AA 205	07AF 206	07B8 207	07C0 208	07C8 209
07CD 210	07DF 211	07ED 212	0805 213	080F 214
081E 215	0821 216	0830 217	0836 218	083D 219
0844 220	084B 221	0852 222	085B 223	0861 224
0869 225	086C 226	087C 227	0885 228	088E 229
0897 230	08A0 231	08A9 232	08B2 233	08BB 234
08C4 235	08CD 236	08DF 237	08E2 238	08EA 239
0956 240	0956 243	095E 244	0966 245	096E 246
0976 247	097E 248	0986 249	098E 250	0996 251
09A3 252	09AA 254	09AD 255	09B3 256	09B8 257
09D5 258	09DE 259	09EE 260	09F7 261	0A00 262
0A09 263	0A12 264	0A1A 265	0A20 266	0A2F 267
0A38 269	0A48 270	0A4D 271	0A4D 272	0A50 273
0A54 274	0A57 275	0A57 276	0A5E 278	0A61 279
0A67 280	0A77 281	0A80 282	0A89 283	0A92 284
0A98 285	0A98 286	0A9F 288	0AA2 289	0AAB 290
0AB1 291	0AB6 292	0ABE 293	0AC7 295	0AD0 296
0AD5 297	0AD5 298	0AD5 299	0ADC 301	0ADF 302
0AE5 303	0AED 304	0AF6 305	0AFC 306	0AFC 307
0B03 309	0B06 310	0B0C 311	0B14 312	0B1D 313
0B23 314	0B23 315	0B26 316	0B27 317	0B27 319
0B2C 320	0B31 321	0B36 322	0B44 323	0B4F 324
0B56 325	0B5B 326	0B73 327	0B80 328	0B89 329
0B8E 330	0B96 331	0B9B 332	0BA9 333	0BBA 334
0BBF 335	0BC6 336	0BD4 337	0BD8 338	0BDB 339
0BE9 340	0BFC 341	0C03 342	0C04 343	0C04 346
0C1C 347	0C21 348	0C26 349	0C33 350	0C43 351
0C4B 352	0C4B 353	0C50 354	0C58 355	0C5D 356
0C6A 357	0C72 358	0C77 359	0C81 360	0C84 361
0C84 362	0C89 363	0C91 364	0C99 365	0C9E 366
0CB5 367	0CBC 368	0CBF 369	0CC8 371	0CCD 372

COPYRIGHT ©1982

DIGITAL RESEARCH

P.O. BOX 579

PACIFIC GROVE, CA 93950

SER. # _____

0CD4 373	0CD4 374	0CE1 375	0CF9 376	0CFE 377
0D06 378	0D08 379	0D13 380	0D2B 381	0D30 382
0D40 381	0D4D 385	0D65 386	0D6A 387	0D72 388
0D82 389	0D87 390	0D87 391	0D8F 392	0D97 393
0DA2 394	0DAA 395	0DB2 396	0DBD 397	0DC5 398
0DCE 399	0DD5 401	0DDD 402	0DE5 403	0DEC 404
0DEF 405	0DF7 406	0DFC 407	0E13 408	0E18 409
0E1B 410	0E1B 411	0E20 412	0E25 413	0E2D 415
0E3A 416	0E42 417	0E4A 418	0E52 419	0E57 420
0E57 421	0E60 422	0E69 423	0E6C 424	0E6C 425
0E76 426	0E7E 428	0E8B 429	0E93 430	0E9B 431
0EA3 432	0EA8 433	0EA8 434	0EB0 435	0EB7 436
0EC2 437	0ECA 438	0ED3 439	0ED6 440	0ED6 441
0EE3 442	0EFB 443	0F00 444	0F08 446	0F0D 447
0F14 448	0F1B 449	0F1E 451	0F23 452	0F2B 453
0F33 454	0F3C 455	0F3C 456	0F3F 457	0F47 458
0F4F 459	0F4F 460	0F54 461	0F5B 462	0F62 463
0F65 464	0F65 465	0F6A 466	0F72 467	0F7A 468
0F83 469	0F86 470	0F8E 471	0F96 472	0FAE 473
0FB1 474	0FB2 475	0FB8 478	0FCF 479	0FD6 480
0FD9 481	0FE2 482	0FE5 483	0FE8 484	03F8 485
03FE 486	0416 488	041C 489	0424 490	0424 491
042E 493	0431 494	0439 495	0439 496	043E 497
044D 498	0451 499	0454 500	0465 501	046B 502
0470 503	047B 504	0487 506	048E 507	0491 508
0494 510	0497 511	049F 512	04A2 513	04A2 514
04A5 515	04AD 517			

COPYRIGHT © 1982
 DIGITAL RESEARCH
 P.O. BOX 579
 PACIFIC GROVE, CA 93950
 SER. # _____

COPYRIGHT ©1982
DIGITAL RESEARCH
P.O. BOX 579
PACIFIC GROVE, CA 93950

SER. # _____